Escapy2.0 Engine
Specification and User Guide

Генрих Тимур Домагальски

31.07.2017 Издаине 1

Содержание

О движке

Escapy2 это игровой движок написанный на java с использованием библиотек Dagger1, libGDX и Gson. Поскольку libGDX является лишь низкоуровневой оберткой над lwjgl - движок дает полноту простора в использовании openGL, в свою очередь Dagger делает код более модульным и масштабируемым. На момент издания этого документа, движок состоит из пяти ключевых пакетов:

1. Context
2. Desktop
3. Graphic
4. Group
5. Utils

Каждому из вышеперечисленных пакетов посвящена отдельная глава, подробнее со структурой api можно ознакомится через javadoc. На конец следует сразу заметить, что данный документ, так же как и сам движок расчитан на разработчиков неплохо знакомых с java и ООП, а так же основами openGL. Основной задачей документа не является скурпулезное описание API - за этим следует идти в javadoc, основная же цель документа - в первую очередь обрисовать возможности движка, его принцип работы, а так же life-cycle и тп.

1 | Начало работы

Вход производится похожим образом как в libGDX и lwjgl - в main с созданием инстанции **LwjglApplication**. Для этого создается объект **LwjglApplicationConfiguration** который загружается из json файла с помощью **EscapyDesktopConfigLoader**, о самих загрузчиках и механизме сериализации в движке более подробно потом.

```
import ...

public class DesktopLauncher {

    public static void main (String[] arg) throws Exception {

        LwjglApplicationConfiguration config = DesktopConfigLoaderBuilder.Default()
                .setLoadedClass(LwjglApplicationConfiguration.class)
                .setSerializedClass(SerializedDesktopConfig.class)
                .setPath(System.getProperty("user.dir") + separator + "Configuration.json")
                .setName("MainConfiguration")
                .build()
        .loadDesktopConfig();

        new LwjglApplication(new EscapyApplicationAdapter(MainEnvironment.class, new MainModule()), config);
    }
}
```

```
{
    "type": "EscapyConfiguration",
    "name": "MainConfiguration",

    "resizable": false,
    "vSync": true,
    "fullscreen": false,
    "forceExit": true,
    "useGL30": true,

    "scrWidth": 1920,
    "scrHeight": 1080,

    "fps": 25
}
```

При создании **LwjglApplication** в качестве аргумента передается **EscapyApplicationAdapter**, который в свою очередь в качестве аргумента принимает класс наследующийся от **EscapyGameContext** и varargs модулей Dagger'a.

```
import ...

public class MainEnvironment extends EscapyGameContext {

    private final EscapyScreen initialScreen;

    @Inject
    protected MainEnvironment(
            Collection<EscapyScreen> escapyScreens,
            EscapyScreen initialScreen,
            EscapyGameContextConfiguration contextConfiguration) {

        super(escapyScreens, contextConfiguration);
        this.initialScreen = initialScreen;
    }

    @Override
    protected EscapyScreen getInitialScreen() { return initialScreen; }

}
```

Подробнее о том как использовать модули Dagger'a можно прочитать на оффициальном сайте проекта (*http://square.github.io/dagger/*). **EscapyGameContext** имеет два конструктора, один из них как аргумент принимает инстанцию класса унаследованного от **EscapyGameContextConfiguration** - абстрактного класса предоставляющего конфигурацию проекта через методы которые можно перегрузить в случае необходимости.

```
import ...

public class MainConfiguration extends EscapyGameContextConfiguration {

    @Override
    public String getResourcesDir() {
        String local = System.getProperty("user.dir").replace( target: separator+"core"+separator+"assets",  replacement: "");
        return local + separator + "res";
    }

    @Override
    public String getConfigsFilePath() { return this.getResourcesDir() + separator + "configurations"; }

    @Override
    public void configurePropertyKeys(PropertyKeysStorage propertyKeyStorage) {
        propertyKeyStorage
                .addPropertyKey("BLEND_SHADERS_ROOT_DIR_PATH")
                .addPropertyValue(getResourcesDir() + separator + "shaders" + separator + "blend")
        .save();
    }
}
```

2 | Context

Самый главный и значимый пакет движка в плане его архитектуры. Его основными элементами являются два субпакета - ***game*** и ***annotation*** и класс ***EscapyGameContext***. Последний наследуется от интерфейса ***EscapyScreenContext*** позволяя тем самым на работу с экранами (сценами).

2.1 Game

Основные эллементы данного субпакета это классы:

- EscapyGameContextConfiguration - абстрактный класс делегирующий настройки
- EscapyScreenContext - интерфейс управления экранами
- EscapyScreen - интерфейс экрана (сцены).
- PropertyKeysStorage - интерфейс позволяет сохранять пары ключ-объект.
- Escapy - синглетон хранящий некоторые настройки.

EscapyScreen

Отдельного внимания заслуживает этот интерфейс, он в свою очередь наследуется от интерфейса ***Screen*** из библиотеки libGDX и содержит callback методы в которых должна находиться логика игры. Класс реализующий данный интерфейс, может (опционально) быть отмечен аннотацией ***@SreenName("...")***, в таком случае этому экрану будет присвоенно имя с помощью которого к этому экрану можно будет обращаться через методы интерфейса ***EscapyScreenContext***.

2.2 Annotation

Содержит аннотации такие как ***@SreenName("...")***, а так же субпакет ***meta*** содержащий процессор аннотаций построенный по шаблону «Декоратор». Если интересуют подробности или возникло желание написать свою собственную имплементацию, то лучшем решением будет отсылка в javadoc или исходники.

3 | Utils

Пакет со вспомогательными классами и прочими полезными вещами. Особого внимания заслуживают:

- ***EscapyArray* и *EscapyAssociatedArray*** - интерфейсы (и их реализации) наследующие Iterable с массивом внутри.
- Пакет ***proxy*** - позволяет инстацировать объекты с listener'ами внутри.
- ***EscapyInstanceLoader*** - позволяет инстанцировать объекты по имени с помощью аннотации ***@EscapyInstanced("...")*** или по имени метода.
- ***EscapySerialized* и *EscapySimpleSerialized*** - интерфейс и абстрактный класс реализующий этот интерфейс, служат шаблоном для сериализуемы с помщью Gson'а классов.

3.1 EscapySimpleSerialized

Так выглядит этот шаблон в исходниках.

```java
import ...

public abstract class EscapySimpleSerialized implements EscapySerialized {

    @SerializedName("type") @Expose public String type = "";
    @SerializedName("name") @Expose public String name = "";
    @SerializedName("attributes") @Expose public List<String> attributes = new LinkedList<>();

    @Override public Collection<String> getAttributes() {
        return attributes;
    }

    @Override public String getName() {
        return name;
    }

    @Override public String getType() {
        return type;
    }

}
```

А так выглядит его json.

```
{
    "name": "",
    "type": "",
    "attributes": ["", "", ""]
}
```

Поскольку все классы движка должны сериализовываться через этот шаблон, код выше является необходимым минумом, для того что бы загрузчики движка могли успешно выполнить свою работу.

3.2 EscapyInstanceLoader

Класс реализующий этот интерфейс позволяет на вызов инстанцирующих методов по имени либо самого метода, либо указанного в аннотации которым этот метод отмечен. Этот механизм очень удобен в использовании в загрузчиках движка и потому повсеместно там используется - для инстанцирования объектов по имени указаноому в json файле, либо для загрузки атрибутов для уже существующего объекта.